Practical Work 4
Ivy/*Rejeu*

This practical work session is dedicated to learning how to use the *Ivy* software bus and the *Rejeu* air traffic simulator in *Java/JavaFX* in order to allow you to integrate these two tools in your transversal project.

duration: 2h

Objectives:

- Use regular expressions,
- Introduce event-driven programming and use a simple textual software bus (*Ivy*) to receive data coming from another application (the air traffic simulator *Rejeu*),
- Use a queue to cleanly access a *JavaFX* application thread from another thread.

Documentation:

Ivy software bus
https://www.eei.cena.fr/products/ivy/

Ivy Javadoc
https://www.eei.cena.fr/products/ivy/documentation/ivy-java-api/index.html?fr/dgac/ivy/package-summary.html

Regular expressions for *Ivy* (see appendix 1)

Rejeu documentation (short introduction in appendix 2 + complete doc provided with Exercise 3 code)

Exercise 1: First communication on *Ivy* bus

Ivy is a software bus, i.e. a distributed system allowing to interconnect on the same network several software components that can be developed in different technologies and can run on different machines. *Ivy* was designed by french civil aviation developers in order to facilitate prototyping of interactive systems, mainly ATC systems, but it's open source and can be used for any textual data exchange.

The communication principle is as follows: each software component (further referred to as a *client*) can send text messages on the bus, and subscribe to messages whose characteristics are specified by a *regular expression* (further referred to as a *regexp*, whose syntax and examples are available in the appendix). Subsequently, when a message corresponding to a specific *regexp* is sent on the bus, the *clients* that subscribed to this *regexp* are notified (you can see it as an event, even if it's not a *JavaFX* `Event`), which allows them to react.

Let's return to the term "distributed" mentioned in the first paragraph. It means that *Ivy* is not centralized, i.e. that there is no *server* but just *clients* directly connected to each other when they connect to the bus and subscribe to *regexps*.

For your very first contact with *Ivy* you will start by connecting two instances of an agent called *IvyGUIMonitor* that allows to send and subscribe to messages through a small *Graphical User Interface:*

- Open the "*Exercise 1\IvyGUIMonitor*" folder retrieved from e-campus,
- Read the "*readme.txt*" file and launch two instances of the *IvyGUIMonitor* application,
- In each instance connect to the default domain (click on the Connect button),
- In the first instance subscribe to the regexp ^Hello (.*) (enter the regexp then click the Bind button),
- From the second instance send the message Hello Man!

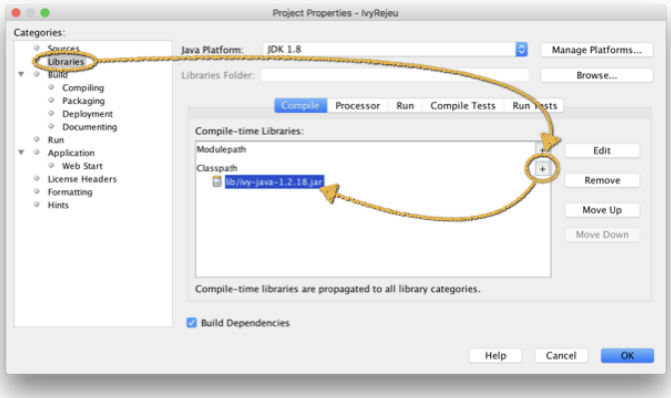
What does the first instance receive (the word(s) in square brackets at the end of the line MSG RECEIVED)? What's that for?

Exercise 2: Using the *Ivy* bus in *Java*

2.1 Create a new "*Java With Ant / Java Application"* project. Retrieve the files for exercise 2 on e-campus and copy the contents of the src folder into your project's src folder. What's up?

2.2 To solve this problem you need to import the "*ivy-java-1.2.18.jar*" library into the project:

- Add a folder named "*lib*" in your project folder and save the library in this folder,
- Right-click on the project name then select the item "*Properties*",
- In the left column select the category "*Libraries*",
- In the tab "*Compile*" press the button + at the right of "*classpath*", choose the option "*Add JAR/ Folder*",
- Select the library "*ivy-java-1.2.18.jar*". Make sure that the library is referenced with its relative path ("*reference as relative path*" option), which will not break your project if you move it later.



This library contains the Java code needed to create an *Ivy* client. You will use it during the rest of this work session and during your project.

The *Ivy* Java client runs in a thread (an independent process inside your application). Make sure to terminate this thread by invoking its `stop()` method when you exit your Java application. In the provided code you will find how to do this in reaction to a message sent on the bus. Test it by sending this message from an `IvyGUIMonitor` instance.

2.3 Using the `Ivy` class Javadoc, modify the code to react to messages like "Hello *FIRSTNAME1*, my name is *FIRSTNAME2*" and return "Nice to meet you *FIRSTNAME2*!", where *FIRSTNAME1* and *FIRSTNAME2* will be two simple regular expressions describing a name.

Test your program with *IvyGUIMonitor* (remember to subscribe to the answer on the *IvyGUIMonitor* side or you won't receive it).

Exercise 3: Retrieving data from *Rejeu*

3.1 We are now going to use *Rejeu* to retrieve air traffic data:

- Read the introduction to *Rejeu* (appendix 2),
- Open the "*Exercise 3\Rejeu*" folder retrieved from e-campus,
- Read the "*readme.txt*" file and follow the installation procedure,
- Launch a *Rejeu* instance (instructions also in "*readme.txt*"),
- Test it with *IvyGUIMonitor* (remember to subscribe to (.*) in order to receive all the messages sent on the bus).

3.2 Study the *Rejeu* documentation (last section of appendix 2) to determine a minimum set of messages to listen to in order to follow the evolution of the traffic.

In your project you will also have to manage safety aspects. In particular you will have to make sure that the navigation orders sent are taken into account by your drone (simulated by *Rejeu*). So you will

probably have other messages to listen to than the simple periodic messages indicating the position of your drone.

3.3 The objective of this exercise is to create an application that listens to *Rejeu* messages on the bus and displays the traffic information periodically emitted by *Rejeu*.

You will have to use the provided code and the `Ivy` class javadoc to write the regular expressions which allow to retrieve the data. To start architecturing your code you will use the provided main class `Ex3Question3_IvyRejeuApplication` and you will create a `communication` package with a class `Ex3Question3_CommunicationManager` in order to factorize all the communication management (it must not remain in the main class).

In your project, you will first be able to reuse this work. But *Rejeu* will be there only to simulate your drone navigation. So this part of the communication management will be incorporated to your *onboard system*. And you will have to add a communication in *APDLC* format between your *onboard system* and your *ground system*: every agent will communicate through the same bus, but will use different messages/formats (*Rejeu* format between *Rejeu* and your *onboard system*, *APDLC* format between your *onboard system* and your *ground system*).

3.4 *Ivy* and the thread of a *JavaFX* application:

For this question you will work with the provided main class `Ex3Question4_IvyRejeuJavaFX`, which implements a simple *JavaFX* application (you will soon learn to do better during the *Rich Graphical Interface Programming* course). Your job here is to modify your communication manager (or create a new one) to write the received data into the `TextArea` of the main class.

Test your modification by starting the main class. Normally errors occur as you receive data from *Rejeu*. These errors are due to the fact that the thread in which the *Ivy* listeners are running is trying to access the *JavaFX* application thread (it tries to modify the `TextArea` content) in a rude way.

Remove this problem by asking politely: actions requested from the *JavaFX* thread should be queued to be executed when the *JavaFX* thread considers it appropriate. To do so you will use the method `javafx.application.Platform.runLater()` (look into the javadoc).

In your project, as in any other well architected code it is not a good practice to access to your *GUI* directly from `CommunicationManager`. So you will soon learn to use one or several data model classes which you will observe from your *GUI* by using *JavaFX* suitable mechanisms.

3.5 Leave cleanly:

When you have finished your simulation close the *JavaFX* window. This does not end the Java application because although the *JavaFX* thread is interrupted by closing the window, the *Ivy* thread has not received any instruction to finish. Therefore it continues listening in the background. To fix this problem:

- Stop your application from *NetBeans* with the button "*Stop*" on the left of the console (the red one),
- Redefine the method `stop()` of your `Application` (this method is called by the *JavaFX* thread when windows are closed),
- From this method call the method `stop()` on the instance of the class `Ivy` used in `CommunicationManager`, which allows you to quit the bus cleanly (as in question 2.2) and, since it is the only thread still alive, to finally quit the application.

Appendix 1: Regular expressions for *Ivy*

A regular expression (*regexp*) is a string of characters describing the structure and content of a text (*patterns*) following a particular syntax. When a pattern is recognized (a *match*), the application can react accordingly.

Several syntaxes exist. Ivy uses *Perl Compatible Regular Expression* (*pcre*).

pcre documentation: http://perldoc.perl.org/perlre.html#Regular-Expressions

A *regexp* is constructed using characters (these will be recognized as they are) and expressions using metacharacters (modifiers, classes and quantifiers) that allow to describe recognizable structures.

Main *modifiers*

^	Match the beginning of the line
.	Match any character (except newline)
$	Match the end of the string (or before newline at the end of the string)
|	Alternation
\	Quote the next metacharacter

Main *classes*

\S	Match a non-whitespace character
\d	Match a decimal digit character
\D	Match a non-digit character
[...]	Match a character according to the rules of the bracketed character class. Example: [a-z] matches "a" or "b" or "c" ... or "z"

Main *quantifiers*

*	Match 0 or more times
+	Match 1 or more times
?	Match 1 or 0 times
{n}	Match exactly n times
{n,}	Match at least n times

A few *regexp* examples

(.*)	match any expression
(\d*)	match a sequence of digits
(\d{4})	match a 4-digit integer
^hello (foo|bar)!$	matches "hello foo!" or "hello bar!" exclusively (no more no less)

In general, the more accurate the *regexp*, the better the performance of the application using them.
regexp can be tested on several websites, for example: http://www.regextester.com

Appendix 2: Introduction to *Rejeu*

***Rejeu* is an air traffic simulator** developed at the study centre of the french Directorate General of Civil Aviation which is used in student projects at ENAC but also in projects for the design of air traffic control tools.

***Rejeu* allows the replay of periods of actual traffic recorded** and possibly modified in order to create particular situations, for example conflicts between several flights. All this data is stored in a text file, with a predefined and fairly flexible format, which is written using external tools from the main traffic and sectorisation formats used within the Directorate General for Civil Aviation. *Rejeu* reads this data, replays the traffic and sends the descriptions of its evolution in the form of text messages on the *Ivy* software bus. All this data is stored in a text file, with a predefined and fairly flexible format, which is written using external tools from the main traffic and sectorisation formats used within the Directorate General for Civil Aviation.

Rejeu reads this data, replays the traffic and sends the descriptions of its evolution as it happens in the form of text messages on the Ivy software bus.

***Rejeu* is also capable of receiving, through the same channel, navigation orders** to modify trajectories in order to test them in real time and then record the whole in a new replay file. The navigation orders are taken into account in a realistic way by calculating a new trajectory based on the characteristics of the flight at the time of the modification but also the performances of the aircraft.

It is these functionalities of replay and modification of trajectories that will be used throughout your project to visualize the position of your drone but also to modify its flight plan.

The messages defined in *Rejeu* can be classified into 4 categories which are explained in the following subsections, taken from the existing documentation.

Messages sent "systematically" by *Rejeu* (events)

They are often related to the time of the simulation and they are similar to events of different nature to which applications subscribe. It is for example the traffic position messages (X, Y, VX, VY, FL...) updated every 8s, for example en route, which are used to refresh the radar image of air traffic controllers.

Some events are not time related, but are a consequence of a change in the data requested by an application. For example, messages indicating that the trajectory of an aircraft changes as a result of a navigation order.

Request for *Rejeu* to make a change (order)

These messages will be sent from an *Ivy* client to ask *Rejeu* to modify its behaviour: to take an action or to take into account modifications to the used data.

For example:

- Set the current simulation time and the desired clock speed,
- Give a navigation order to a plane,
- Save in a file the modifications requested (and made)…

Messages sent to obtain information (queries)

These messages are issued by *Ivy* clients and are similar to requests that require a response from Rejeu.

Examples:

- Request for the list of the flight plan beacons of an aircraft (with time, level...),
- Asking for the current simulation time…

Messages issued by *Rejeu* in response to a query

It is a response on the bus, with an identifier that has been provided by the requester.

Documentation

A french documentation dedicated to *Rejeu* application is provided with Exercise 3 code ("*Exercise 3\Rejeu\doc*" folder).

An english documentation dedicated to *PyRejeu* application (a lite version written in *Python*) is provided with Exercise 3 code ("*Exercise 3\PyRejeu\docs*" folder). The "Supported messages" section should be compatible with legacy *Rejeu* application. *PyRejeu* application itself should be compatible with the use you need in your project but with no guarantee and with a bit more complicated install (dependencies to pull with *pip* tool, so do not use it if you are not used to *pip*).